# Mobile Information Device Programming (11)

Lecturer: Alireza Mousavi
School of Engineering & Design
www.brunel.ac.uk/~emstaam

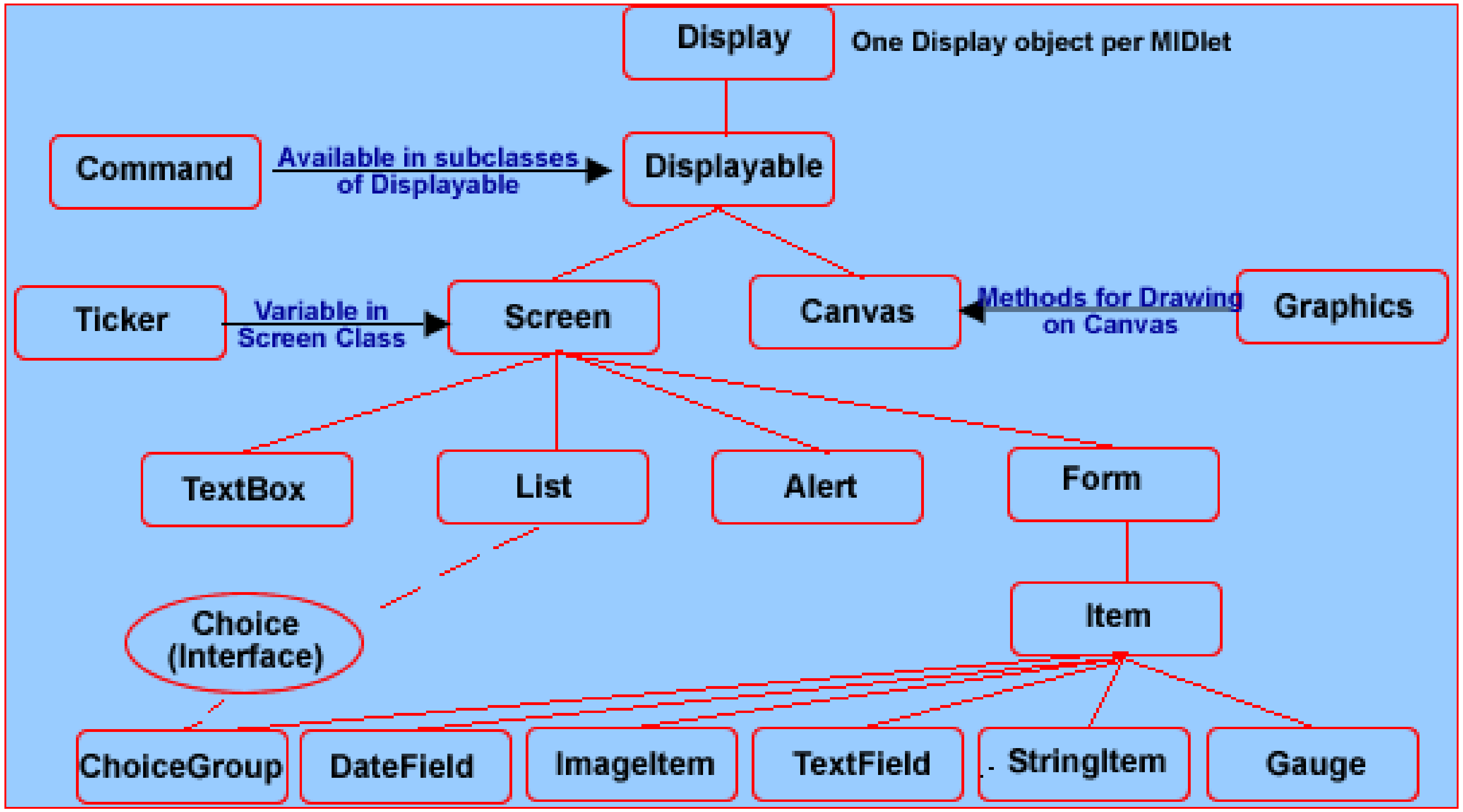
## Lower-Level UI

- Provides free-hand graphical capabilities
- Ideal for specialised applications
- Games

## Main Topics

- Canvas
  - Blank sheet with specific height and width
  - What ever is drawn on it becomes visible
  - Provides methods for low-level event handling
- Graphics
  - **Graphics** are used to draw on canvas
  - Contains methods to draw lines, arcs, rectangles, text
  - Contains methods to control colour and font attributes

# Displayable Class Hierarchy



## Creating Canvas

- You need to first create a subclass of **Canvas**
- Then set it as the current **Displayable**

*Example:*

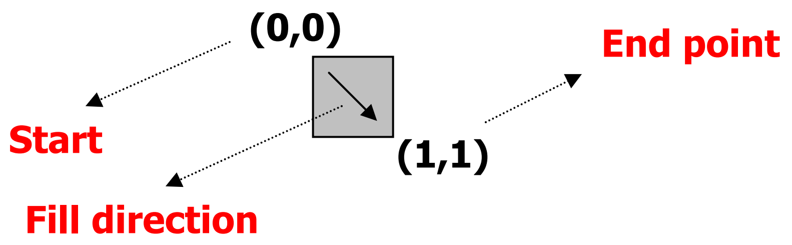
```
class Mycanvas extends Canvas implements CommandListener {
private Command cmExit
…
cmExit = new Command("Exit", Command.EXIT, 1);
addCommand(cmExit)
setCommandListener(this);
…
protected void paint (Graphics g) {
… }
…
Mycanvas canvas = new Mycanvas(this)
Display.setCurrent(canvas);
```

## Origins of Drawing on Canvas

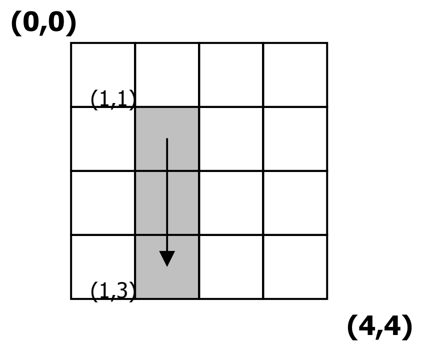
- System Coordinate: Top left-hand corner (0,0)
- The thickness of line or shape is 1 pixel (pen)
- Pixels are 1X1 rectangles
- Filling rule is [right → down ↓ = ↘

# Drawing / Filling Pixels Rule

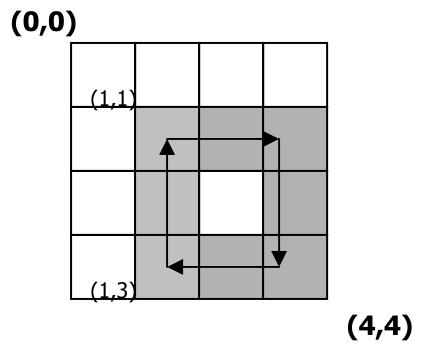
**Example 1: Fill the original point**



**Example 2: Draw a line starting (1,1) and ending (1,3)**



**Example 3: Draw a rectangle from (1,1) to (3,3)**



## Canvas Width & Height

- Methods to query Canvas Width and Height

*int getWidth( )*

*int getHeight( )*

If the height is for example 250 then every Canvas created will have this height

## Painting on Canvas

```
Mycanvas canvas = new Mycanvas(this)
Display.setCurrent(canvas);
```

- This declares a **Canvas** and requests it to be the current displayable
- Just like displaying other components e.g. Form, TextBox, etc.
- But there is a difference

## *paint( )* Method

- **Displayable** class defines *paint( )* abstract method
- Both subclasses **Screen** and **Canvas** implement *paint( )* method
- The *paint( )* method inside **Canvas** is abstract i.e. there is no method body. It is to the subclass to implement this method

```
public abstract class Canvas extends Displayable
protected abstract void paint(Graphics g);
```

- In **Screen** subclass this is different – before leaving *paint( )* the *paintContent( )* method is called

```
public abstract class Screen extends Displayable {
abstract void paintContent(Graphics g);
paint(Graphics g) {
…
paintContent( g); }
```

## Painting Components

- The components on **Canvas** and **Screen** (subclasses of Displayable) are made visible through a call to: *javax.microedition.lcdui.display.setCurrent(Displayable);*

- The difference:
  - **Canvas:** overrides the *paint( )* method and write a code (more control for programmer)
  - **Screen(Form, List, TextBox and Alert):** the *paint( )* and *paintContents( )* methods comprise the code to draw each component

# Canvas Example

The *paint( )* method passes a reference to a **Graphics** object that is used for drawing objects onto the **Canvas**

```
protected void paint(Graphics g){
g.drawString("Hello World", 0, 0, Graphics.TOP | Graphics.LEFT);
g.drawRect(5, 5, 10, 10);
…
}
```

# Some Canvas *paint( )* methods

| ***Method*** | ***Description*** |
|---|---|
| *abstract void paint(Graphics g)* | Draw onto the Canvas using the Graphics Object |
| *final void repaint( )* | Request the canvas to be painted |
| *final void repaint(int x, int y, int width, int height)* | Request that a specific area of the Canvas to be painted |
| *final void serviceRepaints( )* | Immediately process any pending paint requests |
| *boolean is DoubleBuffered( )* | Does the implementation provide double buffering |
| Source: Core J2ME, J. W. Muchow, Sun microsystems | |

# Communication with Application Manager

- When an Application makes a **Canvas** visible it calls the *showNotify( )* method
- The *hideNotify( )* method removes the **Canvas**

| ***Method*** | ***Description*** |
|---|---|
| *void showNotify ( )* | Application manager will be showing the canvas on the display |
| *void hideNotify ( )* | Application manager has removed the canvas from display |

## Intelligent use of showNotify( ) & hideNotify( ) methods

```
protected void showNotify( ) {
 // initialise variable
// start a thread
… }

protected void hideNotify( ){
// reset variables
//stop a thread
…}
```

## Canvas Event handling

There are two ways to interact with Canvas

- Commands
- Low-level Interface (key codes, game actions and pointer events)

## Commands

The four methods available:

- addCommand(*the command*)
- isShown( )
- removeCommand(*the command*)
- setCommandListener(Commandlistener)

**Just like Form, List and TextBox**

## Key Codes

- Key codes are numeric values that map directly to specified keys on a mobile device
- The key codes are guaranteed to be available on any MIDP
- It is normally the standard telephone keypad(0-9,*,#)

```
public static final int KEY_NUM0 = 48;
public static final int KEY_NUM1 = 49;
```

and so on

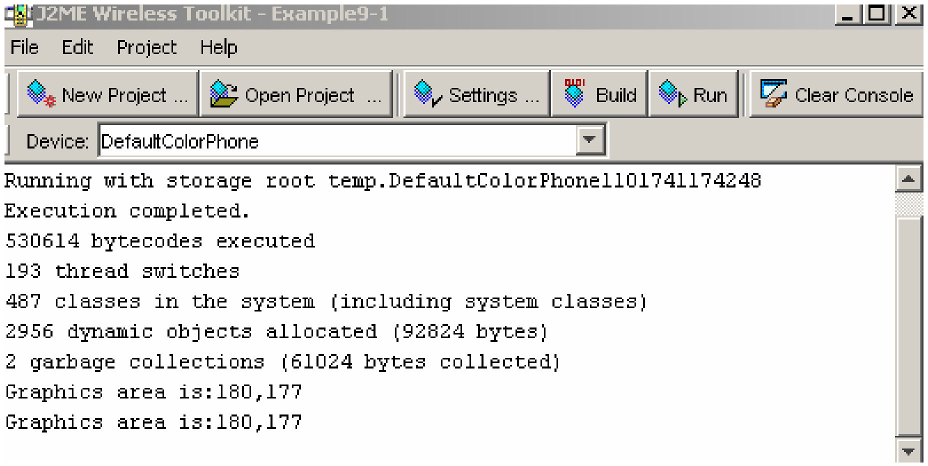
# Key Code methods

| *Method* | *Description* |
|---|---|
| *void keyPressed(int keyCode)* | Invoked when a key is pressed |
| *void keyReleased(int keyCode)* | Invoked when a key is released |
| *void keyRepeated(int keyCode)* | Invoked when a key is repeated(device) |
| *boolean hasRepeatEvents( )* | Does the implementation support repeated keys |
| *String getKeyName(in keyCode)* | Text string representing the key code |
| Source: Core J2ME, J. W. Muchow, Sun microsystems | |

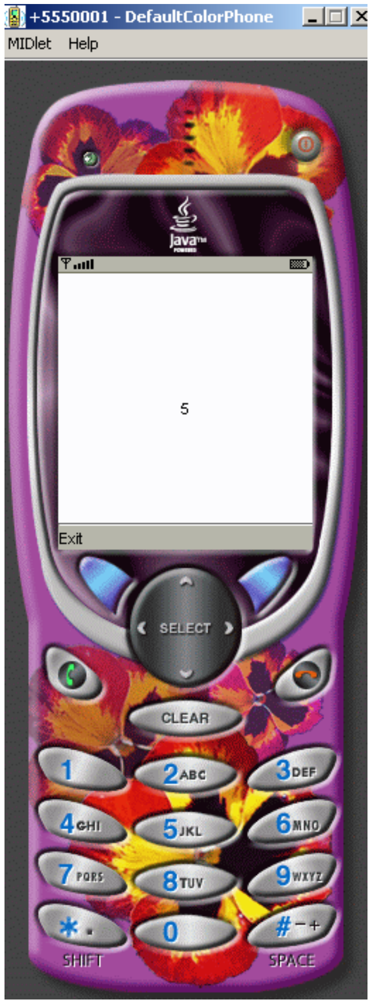
## Example 11-1 key code names

- Create an application that detects commands
- Prints key code name on canvas when the corresponding key is pressed, *keyPressed ( )*
- Send a message to consol showing the graphics area

# Result


J2ME Wireless Toolkit - Example9-1
File Edit Project Help
New Project ... Open Project ... Settings ... Build Run Clear Console
Device: DefaultColorPhone

```
Running with storage root temp.DefaultColorPhone1101741174248
Execution completed.
530614 bytecodes executed
193 thread switches
487 classes in the system (including system classes)
2956 dynamic objects allocated (92824 bytes)
2 garbage collections (61024 bytes collected)
Graphics area is:180,177
Graphics area is:180,177
```



## Notice

There are two methods for managing events:

1. **commandAction( ):** As usual receives *Command* object that generates the event – and the *Displayable* object that the event was received
2. **keyPressed( ):** Is called when a key code creates an event – the code is converted into *String* and a request is made to repaint the *Canvas*

## Game Actions

- Game Actions are defined as set of constants by MIDP
- Each game action is defined as a *static integer*

```
public  static  final  int  UP = 1;
public  static  final  int LEFT= 2;
public static  final int  DOWN = 6;
…
```

- This facilitates event handling
- Each game action will be assigned a key code by the implementation

## Are you Worried!

- Some devices may have specified arrowed keys that actions such as: UP, DOWN, LEFT, RIGHT and FIRE are map to.

- Some devices do not so you
can use 2, 6, 4, 8 and 5
respectively!

## Game Actions

| *Name* | *Description* | *Constant Value* |
|---|---|---|
| UP | Move UP | 1 |
| DOWN | Move DOWN | 6 |
| LEFT | Move LEFT | 2 |
| RIGHT | Move RIGHT | 5 |
| FIRE | FIRE | 8 |
| GAME_A | Custom | 9 |
| GAME_B | Custom | 10 |
| GAME_C | Custom | 11 |
| GAME_D | Custom | 12 |
| Source: Core J2ME, J. W. Muchow, Sun microsystems | | |

## Game Action Methods

| ***Method*** | ***Description*** |
|---|---|
| int getKeyCode(int gameAction) | Define a key code for game action |
| int getGameAction(int keyCode) | Get the game action for the key code – if any |
| String getKeyName(int keyCode) | Get name for a key code |

Example:

*int  keyFire = getKeyCode(FIRE);*
*int  keyLeft = getKeyCode(LEFT);*